```
package server;

import java.io.*;
import util.*;

// System Configuration Information.

public class SystemInfo
{
    // Boolean System Option Bitmasks.

    public final static int         boSelfRegister              =   0x1;    // True if u¬
ser self-registration enabled.
    public final static int         boLogInAndOut               =   0x2;    // True if b¬
adge login and logout enabled.
    public final static int         boDefault                   =   0x2;    // Default o¬
ptions.

    public static SystemInfo        si;
    public int                      iOptions;                               // System op¬
tions bitmask..
    public String                   sAdminPassword;                         // Administr¬
ator password.
    public String                   sCompany;                               // Customer ¬
company name.
    public int                      cDaysToKeepMessages;                    // # of days¬
 to keep messages before deleting.
    public long                     iTimeToSweep;                           // Time of d¬
ay to sweep in millisecs since midnight.
    public MailInfo                 mi;                                     // Mail info¬
rmation.
    public TelephonyInfo            ti;                                     // Telephony¬
 information.
    public UserInfo                 uiDefault;                              // Default U¬
serInfo.
    public int                      iForcedOptions;                         // User opti¬
ons bitmask for options that are forced to system default.
    public int                      iDefaultPermissions;                    // Permissio¬
ns granted to each user by default.
    public int                      iForcedPermissions;                     // User perm¬
issions bitmask for permissions that are forced to system default.
    public StringSet                ssPersonas;                             // Available¬
 genie personas.
    public StringSet                ssRingTones;                            // Available¬
 ring tone file names.
    public boolean                  bForcedPersona;                         // True if p¬
ersona selection is forced to default.
    public boolean                  bForcedRingTone;                        // True if r¬
ingtone selection is forced to default.

    // Default constructor.

    public SystemInfo(ServerConfig sc)
    {
        this();

        iOptions                =   sc.getSystemOptions();
        sAdminPassword          =   sc.getAdminPassword();
        sCompany                =   "";
        cDaysToKeepMessages     =   sc.getDaysToKeepMessages();
        iTimeToSweep            =   sc.getTimeToSweep();
        mi                      =   new MailInfo(sc);
        ti                      =   new TelephonyInfo(sc);
        uiDefault                = new UserInfo(sc);
        iForcedOptions           = sc.getForcedOptions();
        iDefaultPermissions      = sc.getDefaultPermissions();
        iForcedPermissions       = sc.getForcedPermissions();
        bForcedPersona           = sc.isForcedPersona();
        bForcedRingTone          = sc.isForcedRingTone();

    }

    // Copy constructor.

    public SystemInfo(SystemInfo si)
    {
        iOptions                 = si.iOptions;
        sAdminPassword           = si.sAdminPassword;
        sCompany                 = si.sCompany;
        cDaysToKeepMessages      = si.cDaysToKeepMessages;
        iTimeToSweep             = si.iTimeToSweep;
        mi                       = si.mi.copy();
        ti                       = si.ti.copy();
        uiDefault                = si.uiDefault.copy();
        iForcedOptions           = si.iForcedOptions;
        iDefaultPermissions      = si.iDefaultPermissions;
        iForcedPermissions       = si.iForcedPermissions;
        ssPersonas               = si.ssPersonas.copy();
        ssRingTones              = si.ssRingTones.copy();
        bForcedPersona           = si.bForcedPersona;
        bForcedRingTone          = si.bForcedRingTone;
    }

    private SystemInfo()
    {
        findInstalledPersonas();
        findInstalledRingTones();
    }

    // Returns initial UserInfo for creating a user.

    public UserInfo getDefaultUserInfo()
    {
        return uiDefault.copy();
    }

    // Sets array of names of installed genie personas, which are subdirectories of the ¬
prompts subdirectory.
    // (E.g.   "Corporate", "Bombshell", "Stud", "StarFleet")

    private void findInstalledPersonas()
    {
        String[]        saPersonas;

        try
        {
            File    f       = new File(getVoceraHome() + "/data/prompts");
            saPersonas      = f.list();
        }
        catch (Exception e)
        {
            saPersonas      = null;
        }

        Debug.assert(saPersonas != null);
        ssPersonas          =   new StringSet(saPersonas);
        ssPersonas.initCap();
        ssPersonas.remove("Common");
    }

    public static String getVoceraHome()
    {
        return ServerConfig.getVoceraHome();
    }

    // Sets array of names of installed ring tones.

    private void findInstalledRingTones()
    {
        String[]        saRingTones     =   new String[0];

        class Filter implements FileFilter
        {
            public boolean accept(File f)
            {
                return f.getName().endsWith(".wav");
            }
        }

        try
        {
            File    f       =   new File(getRingTonesDir());
            saRingTones     =   f.list();
        }
        catch (Exception e)
        {
            Debug.fail(e);
        }

        ssRingTones         =   new StringSet();

        for (int i = 0; i < saRingTones.length; ++i)
            ssRingTones.add(saRingTones[i].substring(0, saRingTones[i].length() - 4));
    }

    public SystemInfo copy()
    {
        return new SystemInfo(this);
    }

    // Returns directory for downloads of new firmware.

    public String getDownloadDir()
    {
        return getVoceraHome() + "/data/downloads";
    }

    // Returns directory for voice message storage.

    public String getVMessageDir()
    {
        return getVoceraHome() + "/data/messages";
    }

    // Returns directory for user prompts.

    public String getUserDir()
    {
        return getVoceraHome() + "/data/user";
    }

    // Returns directory for location prompts.

    public String getLocationDir()
    {
        return getVoceraHome() + "/data/location";
    }

    // Returns directory for prompts.

    public String getPromptsDir(String sSubdir)
    {
        return getVoceraHome() + "/data/prompts/" + sSubdir;
    }

    // Returns directory for common prompts, such as beeps.

    public String getCommonPromptsDir()
    {
        return getPromptsDir("common");
    }

    // Returns directory for ring tones.

    public String getRingTonesDir()
    {
        return getVoceraHome() + "/data/ringtones";
    }

/*
    // Returns directory for voice enrollment files.

    public String getEnrollmentDir()
    {
        return getVoceraHome() + "/data/enrollment";
    }
*/

    // Returns set of supported personas.

    public StringSet getGeniePersonas()
    {
        return ssPersonas;
    }

    // Returns default genie persona.

    public String getDefaultPersona()
    {
        return uiDefault.o.sGeniePersona;
    }

    // Returns filename of persona introduction .wav file corresponding to persona with ¬
given name.
    // Path of filename is relative to /sounds/ context in web server.

    public String getPersonaIntroductionFileName(String sPersona)
    {
        return getPromptFileName(sPersona, P.persona_introduction);
    }
    // Returns filename of .wav file giving tonal greeting for persona with given name.
    // Path of filename is relative to /sounds/ context in web server.

    public String getGenieEarconFileName(String sPersona)
    {
        return getPromptFileName(sPersona, P.genie_earcon);
    }

    // Returns filename of .wav file giving spoken greeting for persona with given name.
    // Path of filename is relative to /sounds/ context in web server.

    public String getGenieFileName(String sPersona)
    {
        return getPromptFileName(sPersona, P.genie);
    }

    private String getPromptFileName(String sPersona, int iPrompt)
    {
        return "/prompts/" + sPersona + "/" + Prompt.getFileName(iPrompt) + ".wav";
    }

    // Checks that sPersona is among available personas.
    // If so, just returns it, otherwise returns default persona.

    public String checkPersona(String sPersona)
    {
        if (bForcedPersona || !ssPersonas.contains(sPersona))
            return getDefaultPersona();
        else
            return sPersona;
    }

    // Returns set of supported ringtones.

    // Note:  Only the root name is provided.
    // Use getRingToneFilename below to get the fully-qualified file name.

    public StringSet getRingTones()
    {
        return ssRingTones;
    }

    // Returns default ring tone.

    public String getDefaultRingTone()
    {
        return uiDefault.o.sRingTone;
    }

    // Given a ring tone name, returns the corresponding filename.
    // Path of filename is relative to /sounds/ context in web server.

    public String getRingToneFileName(String sRingTone)
    {
        return "/ringtones/" + sRingTone + ".wav";
    }

    // Checks that sRingTone is among available ring tones.
    // If so, just returns it, otherwise returns default ring tone.

    public String checkRingTone(String sRingTone)
    {
        if (bForcedRingTone || !ssRingTones.contains(sRingTone))
            return getDefaultRingTone();
        else
            return sRingTone;
    }

    // Returns prompt path as a function of the given persona.

    public String[] getPromptPath(String sPersona)
    {
        String[]    sPromptPath;

        sPromptPath     =   new String[6];

        sPromptPath[0]  =   getPromptsDir(sPersona);
        sPromptPath[1]  =   getCommonPromptsDir();
        sPromptPath[2]  =   getUserDir();
        sPromptPath[3]  =   getVMessageDir();
        sPromptPath[4]  =   getLocationDir();
        sPromptPath[5]  =   getRingTonesDir();

        return sPromptPath;
    }

    // Returns default prompt path.

    public String[] getDefaultPromptPath()
    {
        return getPromptPath(getDefaultPersona());
    }

    // Returns true if all boolean system-wied options defined by iMask are set.

    public boolean hasOptions(int iMask)
    {
        return (iOptions & iMask) == iMask;
    }

    // Sets/Resets given boolean system-wide options.

    public void setOptions(int iMask, boolean bSet)
    {
        if (bSet)
            iOptions    |=  iMask;
        else
            iOptions    &=  ~iMask;
    }

    // Returns effective user options obtained by combining forced values with nominal v¬
alues given by iUserOptions.

    public int mergeForcedOptions(int iUserOptions)
    {
        return (iUserOptions & ~iForcedOptions) | (iForcedOptions & uiDefault.o.iOptions¬
);
    }

    // Returns true if current default permissions setting satisfies iMask.

    public boolean hasDefaultPermissions(int iMask)
    {
        return (iDefaultPermissions & iMask) == iMask;
    }
    // Returns effective user options obtained by combining forced values with nominal v¬
alues given by iUserOptions.

    public int mergeForcedPermissions(int iPermissions)
    {
        return (iPermissions & ~iForcedPermissions) | (iForcedPermissions & iDefaultPerm¬
issions);
    }

    // Sets/Resets given default permissions.

    public void setDefaultPermissions(int iMask, boolean bSet)
    {
        if (bSet)
            iDefaultPermissions |=  iMask;
        else
            iDefaultPermissions &=  ~iMask;
    }

    // Returns array of filenames for call announcement prompts available to the user.
    // These are found in common prompts directory, nominally set to /bldw/data/prompts/¬
common.
    // For example, array may contain "/bldw/data/prompts/common/foo.wav".

    public String[] getPromptFileNames()
    {
        return WaveReader.listNonSystem(getDownloadDir() + '/');
    }

    // Writes SystemInfo to a stream.

    public void write(DataOutputStream s) throws IOException
    {
        s.writeInt(iOptions);
        s.writeUTF(sAdminPassword);
        s.writeUTF(sCompany);
        s.writeInt(cDaysToKeepMessages);
        s.writeLong(iTimeToSweep);
        mi.write(s);
        ti.write(s);
        uiDefault.write(s);
        s.writeInt(iForcedOptions);
        s.writeInt(iDefaultPermissions);
        s.writeInt(iForcedPermissions);
        s.writeBoolean(bForcedPersona);
        s.writeBoolean(bForcedRingTone);
    }

    // Reads SystemInfo from a stream.

    public static SystemInfo read(DataInputStream s) throws IOException
    {
        SystemInfo      si      =   new SystemInfo();

        si.iOptions                 =   s.readInt();
        si.sAdminPassword           =   s.readUTF();
        si.sCompany                 =   s.readUTF();
        si.cDaysToKeepMessages      =   s.readInt();
        si.iTimeToSweep             =   s.readLong();
        si.mi                       =   MailInfo.read(s);
        si.ti                       =   TelephonyInfo.read(s);
        si.uiDefault                =   UserInfo.read(s);
        si.iForcedOptions           =    s.readInt();
        si.iDefaultPermissions      =    s.readInt();
        si.iForcedPermissions       =    s.readInt();
        si.bForcedPersona           =    s.readBoolean();
        si.bForcedRingTone          =    s.readBoolean();

        return si;
    }

    // Telephony integration.

    public static class TelephonyInfo
    {
        public boolean              bEnabled;                          // True if telep¬
hony enabled.
        public String               sAreaCode;                         // Local area co¬
de.
        public String               sLocalAccess;                      // Line access s¬
equence for local calls.
        public String               sLongDistanceAccess;               // Line access s¬
equence for calls in which area code must be supplilled.
        public String               sVoiceMailAccess;                  // Sequence to g¬
et into voice mail.

        public TelephonyInfo(ServerConfig sc)
        {
            bEnabled                =      sc.isTelephonyEnabled();
            sAreaCode               =      sc.getAreaCode();
            sLocalAccess            =      sc.getLocalAccess();
            sLongDistanceAccess     =      sc.getLongDistanceAccess();
            sVoiceMailAccess        =      sc.getVoiceMailAccess();
        }

        // Copy constructor.

        public TelephonyInfo(TelephonyInfo ti)
        {
            bEnabled            =   ti.bEnabled;
            sAreaCode           =   ti.sAreaCode;
            sLocalAccess        =   ti.sLocalAccess;
            sLongDistanceAccess =   ti.sLongDistanceAccess;
            sVoiceMailAccess    =   ti.sVoiceMailAccess;
        }


        public TelephonyInfo()
        {
        }

        public TelephonyInfo copy()
        {
            return new TelephonyInfo(this);
        }

        public static TelephonyInfo read(DataInputStream s) throws IOException
        {
            TelephonyInfo   ti      =   new TelephonyInfo();

            ti.bEnabled             =   s.readBoolean();
            ti.sAreaCode            =   s.readUTF();
            ti.sLocalAccess         =   s.readUTF();
            ti.sLongDistanceAccess  =   s.readUTF();
            ti.sVoiceMailAccess     =   s.readUTF();
            return ti;
        }

        public void write(DataOutputStream s) throws IOException
        {
            s.writeBoolean(bEnabled);
            s.writeUTF(sAreaCode);
            s.writeUTF(sLocalAccess);
            s.writeUTF(sLongDistanceAccess);
            s.writeUTF(sVoiceMailAccess);
        }
    }

    public static class MailInfo
    {
        // Mail server type.

        public String           sServerType;            // One of "pop3" or ¬
"imap".
        public String           sHost;                  // Mail host.
        public String           sUserName;              // Client username.
        public String           sPassword;              // Client password.
        public String           sSMTPHost;              // SMTP mail host.
        public int              iCheckIntervalMillis;   // Mail check interv¬
al in milliseconds.
        public String           sDefaultRecipient;      // Email address of ¬
default mail recipient.

        // Gets default MailInfo.

        public MailInfo(ServerConfig sc)
        {
            sServerType             =   sc.getMailServerType();
            sHost                   =   sc.getMailHost();
            sUserName               =   sc.getMailUserName();
            sPassword               =   sc.getMailPassword();
            sSMTPHost               =   sc.getMailSMTPHost();
            iCheckIntervalMillis    =   sc.getMailCheckInterval();
            sDefaultRecipient       =   sc.getMailDefaultRecipient();
        }

        // Copy constructor.

        public MailInfo(MailInfo mi)
        {
            sServerType             =   mi.sServerType;
            sHost                   =   mi.sHost;
            sUserName               =   mi.sUserName;
            sPassword               =   mi.sPassword;
            sSMTPHost               =   mi.sSMTPHost;
            iCheckIntervalMillis    =   mi.iCheckIntervalMillis;
            sDefaultRecipient       =   mi.sDefaultRecipient;
        }

        public MailInfo()
        {
        }

        public MailInfo copy()
        {
            return new MailInfo(this);
        }
        public static MailInfo read(DataInputStream s) throws IOException
        {
            MailInfo    mi                =   new MailInfo();

            mi.sServerType                =   s.readUTF();
            mi.sHost                      =   s.readUTF();
            mi.sUserName                  =   s.readUTF();
            mi.sPassword                  =   s.readUTF();
            mi.sSMTPHost                  =   s.readUTF();
            mi.iCheckIntervalMillis       =   s.readInt();
            mi.sDefaultRecipient          =   s.readUTF();

            return mi;
        }

        public void write(DataOutputStream s) throws IOException
        {
            s.writeUTF(sServerType);
            s.writeUTF(sHost);
            s.writeUTF(sUserName);
            s.writeUTF(sPassword);
            s.writeUTF(sSMTPHost);
            s.writeInt(iCheckIntervalMillis);
            s.writeUTF(sDefaultRecipient);
        }
    }

    public void merge(SystemInfo siOld, SystemInfo siBase)
    {
        (new Merger()).merge(this, siOld, siBase);
    }

    // Three-way merger.

    private static class Merger extends ThreeWayMerger
    {
        static String sBitVectors[]     =
        {
            "iOptions", "iDefaultPermissions", "iForcedPermissions", "iPermissions"
        };

        public Merger()
        {
            super(sBitVectors);
        }

        public boolean isAtomic(Class c)
        {
            return super.isAtomic(c) || c == UserInfo.ForwardingInfo.class || c == Buddy¬
Set.class;
        }
    }
}
```